**Rudolph Travers**

**CS-300**

**8/15/2025**

**Project One**

**//Vector - Milestone 1**

//Define the course class

Class Course{

String courseNum //Variable to store the course number

String courseName //Variable to store the name of the course

Vector<string>preReqs //used to store a list of prerequisites for the course

}

// function to ensure there are at least two parameters on each line

bool courseFormatVal(string line){

//need to separate line into its different parameters

Use getline and specify a delimiter to parse info

// USE LOOP TO VERIFY IF LINE HAS 2 0R MORE PARAMETERS

If(parameters.size()<2{

Print “Invalid format”

Return false

}

//open and read the csv data file and return list of lines

Vector<string> readFile(string csvPath){

Init vector<string> courseLines //vector to store the lines read from csvPath

//Open and read file

inputFile.open(string)

if (inputFile.is\_open()

read line from file

add line to courseLines //save the line in the created vector

inputfile.close()

return courseLines

}

//function to create Course objects from lines

Vector<course>processCourses(vector<string> lines){

Init vector<Course>courses

For(line in courseLines){

If(courseFormatVal(line))

Use getline and a delimiter to split line into different components

// make a new course

Init Course newCourse

newCurse.courseNum = param[0]

newCourse.courseName = param[1]

}

}

//function to print details of a course

Void printCourseInfo(vector<Course> courses, string courseNum){

//use loop to cycle through courses until courseNum is found

For(course){

If(course.courseNum == courseNum)

cout << “Course Number: “ + courseId <<

cout << “Course Title: “ + courseName <<

cout << “Prequisites: “ + preReqs <<}

//function to print all courses in alphanumeric order

Function printAllCourses

Sort courses by courseId

For every course in courses vector

cout << “Course Number: “ + courseId <<

cout << “Course Title: “ + courseName <<

cout << “Prequisites: “ + preReqs <<

**//HashTable - Milestone 2**

//Define the course class

Class Course{

String courseNum //Variable to store the course number

String courseName //Variable to store the name of the course

Vector<string>preReqs //used to store a list of prerequisites for the course

}

// function to ensure there are at least two parameters on each line

bool courseFormatVal(string line){

//need to separate line into its different parameters

Use getline and specify a delimiter to parse info

// USE LOOP TO VERIFY IF LINE HAS 2 0R MORE PARAMETERS

If(parameters.size()<2{

Print “Invalid format”

Return false

}

//create hash table to store course objects

class hashtable{

private:

static const int tablesize = ????

struct course{

string courseId;

string courseName;

string preReqs;

}

Node key

//method to insert data into has table

While not at end of file

Split each line into 3 different values

For values(courseId, courseName, preReq)

If node = open

Insert values to the hash table location

Else

Set courseName and preReqs to null

Find next open node

//Function to Print courses

Function printCoursesHashTable(hashTable):

Gather all courses from the hashTable

Sort course in alphanumeric order

For each course

Print “Course Number: “ + course.courseId

Print “Course Title: “ + course.courseName

Print “Prereqs: + course.preReqs

//Function to search for a specific course using unique Id

Function searchCourses

Prompt user to enter uniqueId

If course is found

Print “Course Number: “ + course.courseId

Print “Course Title: “ + course.courseName

Print “Prereqs: + course.preReqs

Else:

Print ”Course could not be found”

**//Binary Search Tree - Milestone 2**

//Define the course class

Class Course{

String courseId //Variable to store the course number

String courseName //Variable to store the name of the course

Vector<string>preReqs //used to store a list of prerequisites for the course

}

// function to ensure there are at least two parameters on each line

bool courseFormatVal(string line){

//need to separate line into its different parameters

Use getline and specify a delimiter to parse info

// USE LOOP TO VERIFY IF LINE HAS 2 0R MORE PARAMETERS

If(parameters.size()<2{

Print “Invalid format”

Return false

}

//Method to load data into the binary search tree

// make bst

BinarySearchTree\* bst;

Bst=new BinarySearchTree();

Course newcourse;

Create way to add new nodes to the BST

If root is null. + root

If < root node add new node to the left

Else new node = left node;

If new node > root node add new node to the right

Else new node = right node

Void insertCourses(csvPath, BinarySearchTree\* bst)

for (unsigned int i = 0; i < file.rowCount(); i++)

bst->insert(course.newCourse);

//Print course Info

Function void printCourseInfo(tree <course> courses, courseId)

Get userInput

Match current node to root node

While current is not null

If course.courseId = current node

Output: course.courseInfo

If coursed is < root

Current = left node

Else

Current = right node

**//Menu Psuedocode**

**Function main():**

**While True:**

**Print “Menu:”**

**Print “1. Load Data”**

**Print “2. Print Courses”**

**Print “3. Search for course”**

**Print “9.Exit”**

**Ask for user input**

**If user input = 1:**

**Call function to load Data from csvPath**

**Else if choice = 2:**

**Call function to Print all the courses in the data structure**

**Else if choice = 3:**

**Ask user to enter the courseId**

**Call function to search for a course**

**Else if choice = 9:**

**End the program**

**Else:**

**Cout << Invalid selection. Try again.” <<**

| **Vector** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **Create the vector** | 1 | 1 | 1 |
| **Create vector line item for each new course** | 1 | n | n |
| **Check if course has a prerequisite** | 1 | n | 1 |
| **If prerequisite exist add it to the course line** | 1 | n | n |
| **Make new course item in the vector** | 1 | n | n |
| **Total Cost** | | | 5n + 1 |
| **Runtime** | | | O(n) |

| **Hash table** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **Create the hash table** | **1** | **1** | **1** |
| **Function to insert courses into hastable** | **0** | **0** | **0** |
| **Make course key** | **1** | **n** | **n** |
| **Assign new node to key** | **1** | **n** | **n** |
| **Assign old node next to null pointer** | **3** | **n** | **n** |
| **Find the next open node** | **1** | **n** | **n** |
| **Add new node to hash table** | **1** | **n** | **n** |
| **Insert course item** | **1** | **n** | **n** |
| **Total Cost** | | | **9n + 1** |
| **Runtime** | | | **O(n)** |

| **Binary Search Tree** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **Function to add node** | **0** | **0** | **0** |
| **Add root node** | **1** | **1** | **1** |
| **If node < root add to the left** | **1** | **n** | **n** |
| **This becomes the left node** | **1** | **n** | **n** |
| **If node > root add to the right** | **1** | **n** | **n** |
| **This becomes the right node** | **1** | **n** | **n** |
| **Insert course item** | **1** | **n** | **n** |
|  |  |  |  |
|  |  |  |  |
| **Total Cost** | | | **6n + 1** |
| **Runtime** | | | **O(n)** |

**Choosing a data structure is a critical design decision when developing a system. Each data structure has its own unique advantages and disadvantages that influence how the system performs. The most important factors to assess when choosing a data structure is the type of data that needs to be managed and the requirements of the system.**

**Vectors are good because each element in the vector has an index so elements in the vector can be looked up quickly. Vectors must search each item line by line so when the entire list needs to be searched it is a less efficient process.**

**Hash tables are good for storing complex data, searching for data is fast because each item has a list. Hash tables make it hard to process large amounts of data tied to a key.**

**Binary trees are good for searching and organizing data. They are also good for linking data together via nodes. They aren’t good for searching an item because in order to traverse the tree you must start at the root node.**